Sebastian Götz, Thomas Kühn, Christian Piechnick, Georg Püschel, Uwe Aßmann

A Models@run.time Approach for Multi-objective Self-optimizing Software

Erstveröffentlichung in / First published in:

ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth, 08. – 10.09.2014. SpringerLink, S. 100 – 109. ISBN 978-3-319-11298-5.

DOI: https://doi.org/10.1007/978-3-319-11298-5_11

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

A Models@run.time Approach for Multi-objective Self-optimizing Software

Sebastian Götz, Thomas Kühn, Christian Piechnick,
Georg Püschel, and Uwe Aßmann

Technische Universität Dresden, Software Technology Group
sebastian.goetz@acm.org,
{thomas.kuehn3,christian.piechnick,georg.pueschel,
uwe.assmann}@tu-dresden.de

Abstract. This paper presents an approach to operate multi-objective self-optimizing software systems based on the models@run.time paradigm. In contrast to existing approaches, which are usually specific to a single or selected set of objectives (e.g., performance and/or reliability), the presented approach is generic in that it allows the software architect to model the relevant concerns of interest to self-optimization. At runtime, these models are interpreted and used to generate optimization problems. To evaluate the applicability of the approach, a scalability analysis is provided, showing the approach's feasibility for at least two objectives.

1 Introduction

A central, required characteristic of future software systems is their ability to adjust themselves to changing environments. Notably, such adjustments fulfill a certain purpose: they reduce or eliminate the deviance of the system from it's desired state. Often, this desired state is an interpretation of *multiple goals* (i.e., objectives) in the context of the current system state. Hence, to compute the required adjustments to reconfigure the system to its desired state, multi-objective optimization (MOO) approaches are required.

Systems capable of reconfiguring themselves with the aim to be operate optimally with regard to specified goals are called self-optimizing systems [1], where the term *self* refers to the ability of the system to autonomously react to changes in the environment. The goals usually span various non-functional properties (NFPs) of the system. For example, performance, energy consumption, reliability and availability to name but a few. The need for MOO approaches arises when multiple, potentially competing, NFPs shall be optimized together.

Two general classes of MOO can be distinguished: a-priori and a-posteriori approaches [2]. The first type tries to unify all objectives into a single objective. Such approaches often use utility-theory, but have the problem that not all NFPs are comparable with each other and, thus, a unification of the objectives is semantically wrong [3]. Hence, a-posteriori approaches, which do not unify the objectives, but are know to be very time-consuming [2], are required.

In our previous work, we proposed [4] a model-driven approach for single-objective self-optimizing systems enabling the developer to model structural

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

and behavioral aspects of the system, and extract runtime information from the system into runtime models used to automatically generate optimization problems for standard problem solvers. The solution presented in this paper extends this approach, to allow for the operation of a-posteriori multi-objective self-optimizating systems.

This paper is structured as follows. The next section introduces a running example. Section 3 briefly outlines our previous work. Section 4 then describes our solution for the operation of multi-objective self-optimizing systems. The approach is evaluated in terms of its scalability in Section 5. Finally, Section 6 covers related work and Section 7 concludes the paper.

2 Running Example: Confidential Sorting

A confidential sort is to be performed whenever the list subject to sorting contains confidential data. This data, in consequence, has to be encrypted. A typical approach to encryption is the use of a pair of public and private keys (e.g., SHA-2 [5]). The bigger the keys used for encryption are, the more effort is required for an attacker to decrypt the data. But, the bigger the keys, the more time is required to encrypt the data. Typically, users intend to get their lists sorted as fast as possible, but encrypted as much as possible. Listing 1.1 shows such a user request for a list with 200.000 elements. The request is formulated against the structure model shown in Figure 1(b). The list, subject to sorting, is encrypted prior to sorting. This way the sort algorithm can be placed on any available machine, even if it is not considered safe (due to its encryption).

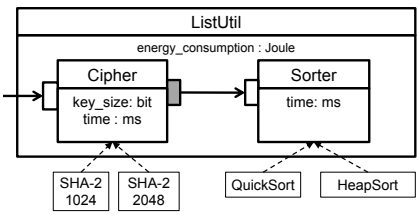
3 Single-objective Self-optimization

In previous work [4], we presented a component-based metamodel and quality contract language to be used by developers of self-optimizing software systems. Moreover, we proposed a model-driven approach [4] utilizing these models to automatically generate optimization problems and, thereby, overcome the limitations of related approaches. This section recapitulates the specifications for the single objective case, which serve as a base for the multi-objective case. Due

Listing 1.1. Confidential Sort Request.

```
1 import ccm [./sort.structure]
2 target platform [./current.variant]
3
4 call ListUtil.sort expecting {
5         list_size = 200000
6         response_time minimize
7         encryption_key_size maximize
8 }
```

(a)



(b) Referenced Structure Model

Fig. 1. Request and Model for the Confidential Sort Example

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

```
1  contract SHA-2 implements Cypher.encrypt {
2    mode secure {
3      requires resource RAM { free_size min: 5 * message_size }
4      requires software RandomGen { value_range min: 100 }
5      provides key_size = 4096
6      provides runtime max: f1(message_size)
7    }
8    mode fast {
9      requires resource RAM { free_size min: 1 * message_size }
10     requires software RandomGen { value_range min: 25 }
11     provides key_size = 256
12     provides runtime max: f2(message_size)
13   }
14 }
```

Listing 1.2. An Example QCL Contract for an Encryption Implementation

to space limitations, we cannot fully recapitulate the approach, but focus on the specification of quality contracts and the generation of optimization problems.

3.1 The Quality Contract Language (QCL)

Implementations and devices can be characterized in terms of their behavior and dependencies to each other by contracts, which are a special type of Quality-of-Service contract [6] and are comprised of assumptions and guarantees on properties defined in structural models. Listing 1.2 depicts an example contract for an encryption implementation and illustrates the mentioned concepts.

QCL contracts define different modes, representing levels of satisfaction for users of the system. For example, a contract for an encryption implementation could define the modes *secure* and *fast*, representing a variant with large encryption key (4096 bit), which will take more time and memory compared to the second variant, which has a small encryption key (512 bit), but will be much faster. For the processing of highly confidential data the first mode will be prefered, whereas for less critical data the second mode is more likely to be prefered.

Each mode specifies a set of assumptions (`requires`). These assumptions are either tight to the properties of the implementation itself or to those of other system constituents. For example, an encryption implementation requires an implementation of a random number generator and has requirements on memory. In addition, each mode defines a set of guarantees for the properties of the respective implementation (`provides`). For example, the *maximum* `response time`, which is characterized as a function, which is replaced at runtime by an empirically derived, approximated function specific to the respective hardware.

3.2 Automatic Generation of Integer Linear Programs

At runtime, whenever a better system configuration for a request is being searched for, the runtime environment generates an integer linear program, which characterizes which system configurations are valid and when they are optimal. The problem solved is the question, *which* software components are required to process the user request and *which* implementations of them running on *which* resources ensure the non-functional requirements of the user best.

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

```
1  /* objective functions */
2  min: 25 x2 + 35 x3 + 180 x0 + 240 x1; //response time
3  max:  0 x2 +  0 x3 + 512 x0 + 4096x1; //encryption key size
4  /* architectural constraints */
5  x0 + x1  = 1;
6  x2 + x3  = 1;
7  /* Resource negotiation */
8  12 x3 + 22 x1 <= 84;
9  12 x2 + 22 x0 <= 48;
10 /* NFP negotiation */
11 97 x2 + 97 x3 >= 52 x0 + 52 x1;
12 /* binary constraint */
13 bin x2, x3, x0, x1;
```

Listing 1.3. Generated ILP for Confidential Sort Example.

Thus, the decisions to be made comprise the selection of implementations and their mapping to resources. Such a problem can be expressed as an integer linear program (ILP), which is comprised of a set of variables, an objective function and a set of constraints. As shown in our previous work, it suffices to encode all possible decisions as variables. For example, the Boolean variable `b#cypher#sha2#server2` denotes the decision to run the SHA-2 implementation of the encryption component on a server called *server2*.

An example ILP for the running example (cf. Sect. 2) is shown in Listing 1.3. The variables x_0 and x_1 represent the decision for the SHA-2 algorithm with a 512 bit key or a 4096 bit key respectively. The variables x_2 and x_3 represent the decision for `QuickSort` or `HeapSort`. To keep the number of variables low, the example comprises only one server.

The first objective function describes the effect of each decision on the overall response time w.r.t. the current user request. It resamples the fact that `QuickSort` is faster than `HeapSort` and encryption with a 512 bit key is faster than with a 4096 bit key. The second objective function shows, explicitly, the effect of using a 1024 or a 2048 bit key and, by this, allows to consider the key size as separate dimension of the Pareto front.

The constraints of the generated ILP denote which combinations of decisions lead to a valid system configuration. There are three types of constraints:

1. Architectural constraints (cf. line 4&5): denote which components are required to serve the user request and the need to select at least one implementation of these components.
2. Negotiation of resources (cf. line 7&8): denote the impact of the decisions on resource properties, which are physically limited (e.g., maximum memory).
3. Negotiation of non-functional properties (cf. line 10): denote the interplay between guarantees and assumptions on NFPs as stated in QCL contracts by each decision.

Finally, the decision variables are constrained to be Boolean (cf. line 13).

In previous work, we showed the applicability of the approach to compute an optimal configuration for a single user objective only. In this paper, we investigate the case of multiple, potentially competing objectives.

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

4 Multi-objective Self-optimization

This section covers our approach to identify a set of Pareto-optimal system configurations using multiple objective integer linear programming (MOILP). This set of configurations is then presented to the user, who can select his preferred alternative. Pareto-optimality denotes that each configuration presented to the user is the best in all but one objective.

The approach to generate MOILP does not deviate from the approach to generate single-objective ILPs except for the generation of multiple objective functions, where each objective function is associated to a non-functional property defined by the user to be of his interest. Integer variables are required to cover clear yes/no decisions. In the following, we show how the resulting MOILP can be efficiently solved.

To determine a Pareto-optimal set of solutions for a MOILP, a variety of approaches have been developed in the past decades. An appropriate MOILP approach has to be able to scale and has to support multiple objectives of free form. We decided to apply the approach introduced by Klein and Hannan in 1982 [7], which fulfills these requirements [8]. As basis for explaining their approach, firstly 0-1 MOLIPs are defined as shown in Equation 1. C is a (p,n)-matrix covering p objective functions over n decision variables denoted by the vector x of size n. A is an (m, n)-matrix covering the left hand side of m constraints over n variables. The vector b of size m denotes the right hand side of these constraints.

$$min\{Cx : Ax \geq b, x \geq 0, x \in \mathbb{B}^n\}$$

In each iteration of the approach, a set of single objective ILPs is to be solved. The ILPs of iteration i are denoted by P_i. As a starting point P_0 is to be derived. Here one of the p objective functions is randomly selected denoted by $0 \leq s \leq p$. All other objective functions are omitted from P_0. This leads to a single-objective ILP, which can be handled by standard ILP solvers like LP Solve [9]. The solution of P_0 is the first solution being part of the Pareto front and used to compute the succeeding ILPs.

In general the construction of ILPs of P_i for $i > 0$ depends on the solutions found until solving the last ILP (i.e., P_{i-1}). The variable r denotes the number of solutions found until P_{i-1}. Klein and Hannan formalized the construction as follows:

$$min : z_s = c^s x$$
$$subject\ to : Ax \geq b, x \geq 0, x \in \mathbb{B}^n \quad and \quad \bigcap_{i=1}^{r} \bigcup_{k=1 \wedge k \neq s}^{p} c^k x \leq c^k y^i - f_k$$

Thus, all ILPs to be solved have a single objective function, namely, the objective function selected for P_0. The additional constraints added to the ILP depend on the number of solutions found until P_{i-1}, denoted by r, and the number of objective functions excluding objective function s selected in P_0. For each objective function, except for s, a disjunctive set of constraints is added. Additionally, for

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

each found solution these disjunctive sets of constraints are conjunctivly added to the ILP. The constraint term $c^k x \leq c^k y^i - f_k$ describes that the k^{th} objective function has to have a value lower or equal to the value of the k^{th} objective function for solution y^i minus a constant f_k. If this constant is set to $f_k = 1$ the approach is guaranteed to provide the complete Pareto front [7, p. 380]. For $f_k > 1$ a subset of the Pareto front is determined.

A crucial detail of the approach is the use of the operators $\cup$ and $\cap$. Standard ILP solvers implicitly assume a conjunction over all constraints. Disjunction ($\cup$-operator) is not supported, but can be handled by solving multiple alternative ILPs instead. This practically leads to a combinatorial explosion of ILPs to be solved. As in Klein and Hannan's approach all disjunctions have the same size $(p-1)$, the number of ILPs to be solved in P_i can be computed as r^{p-1}.

In summary, the basic principle of Klein and Hannan's approach is to successively exclude solutions by enriching an initial ILP with constraints. The original objective functions are used to iteratively restrict the feasible area of an initially selected single-objective ILP. In each iteration the solutions found are added to the Pareto front. The algorithm terminates when all ILPs of P_i are infeasible, i.e., no additional solutions are found. Applying this approach to compute an optimal system configuration for a given user request deviates from the approach for a single objective function only in the number of solutions presented to the user, who has to select the most suitable solution from the Pareto front.

5 Evaluation

This section evaluates the application of Klein and Hannan's approach to MOILP for the computation of an optimal system configuration.

5.1 Approach and Methodology

We randomly generated $C \times S$ pipe-and-filter systems capturing C component types to be mapped on S servers, where each component type has 2 implementations. Due to the combinatorial explosion of ILPs to be solved only 2×2 to 30×30 systems have been measured for systems with more than 2 objectives. The bi-objective case has been evaluated against systems up to 100×100. We refer the interested reader to [4] for the single objective case.

Measurements were taken for 841 systems with 3 and 4 objective functions to analyze the impact of a growing number of objectives. All of these measurements have been conducted on a DELL Alienware X51, Intel i7-2600, 8 GB RAM, 64bit Windows 7 with LP Solve 5.5.20. A notable fact about the special case of two objective functions is the lack of *or-constraint*-blocks in the generated ILPs of P_i where $i > 0$. This is because the number of constraints per or-block equals the number of objective functions minus one, i.e., one in the case of two objective functions. In consequence, there is no combinatorial explosion of ILPs to be solved. Instead, as many ILPs are to be solved as solutions are found (and a final ILP, which is infeasible).

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

As complex MOILPs can require several days to be solved, the measurements taken for MOILPs were limited to 2 minutes for practical reasons. Due to the combinatorial explosion of ILPs to be solved, MOILPs with more than 2 objective functions, a second reason for failing is to be considered: lack of memory. For all measurements 4 GB of main memory are reserved. But, as will be shown in the following, this does not suffice for MOILPs having a big Pareto front. Thus, a MOILP can fail by timeout or by running out of memory.

The generation time of MOILPs is similar to the single objective case [4], because the generation process is almost the same, except for the generation of multiple objective functions. Hence, we do not separately analyze the generation time, but focus on the overall runtime of the approach.

5.2 Results

Figure 2(a) shows the runtime for 2x2 to 100x100 systems with 2 objective functions. As can be seen, the number of servers has a stronger impact than the number of components. The runtime grows very fast per server (approximately 60 seconds already at 5 servers), but slower per additional component (below 1 second until 10 components). Only 27,12% of all MOILPs timed out (2658 of 9801). Notably, no bi-objective MOILP run out of memory. The mean runtime across all systems showed up to be at 36,54 s, the median runtime at 60,12 s only. The 3rd quantile is at 62,92s.

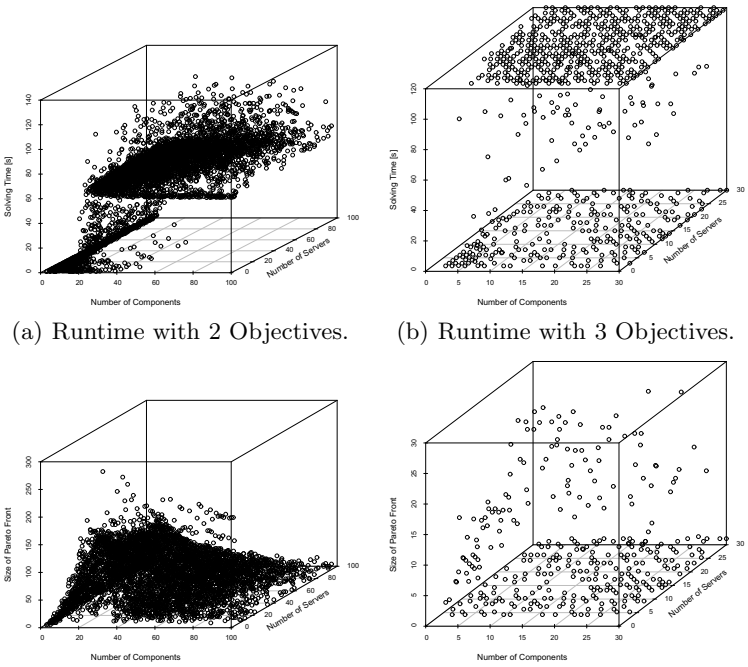
Two further properties are of interest to interpret the behavior of the MOILP solver: the number of ILPs generated for each MOILP and the size of their Pareto front. Figure 2(c) depicts the correlation between the number of components and servers to the size of the Pareto front. For the bi-objective case, the size of the Pareto front almost equals the number of solved ILPs.

Another insight from Figure 2(c) is the high number of solutions in the Pareto front for relatively small systems. For systems as small as 10x10 more than 50 solutions are part of the Pareto front. This poses a challenge to user interaction. The question is how to present this high amount of solutions to the user so he can make an educated decision, but this is out of this paper's scope.

An investigation of 3 objective MOILPs shows, as to expect, considerably worse performance. In each iteration an additional or-block of two constraints is added, leading to a quadratic increase in ILPs to be solved. With a mean runtime of 79,391 s and its third quantile at the timeout limit of 2 minutes its more than 4 times worse than the bi-objective measurements. In contrast to the bi-objective case many more MOILPs timed out and ran out of memory. In total 305 MOILPs (36,27%) timed out and 195 MOILPs (23,19%) ran out of memory, where both sets do not overlap. Hence, for 500 MOILPs (59.45%) no solution could be found. Interestingly, these ≈60% of failed MOILPs are not concentrated on bigger systems, but range from small to big systems as can be seen in Figure 2(b).

Moreover, Figure 2(b) shows that most MOILPs, which do not timeout or run out of memory are solved very fast as there are more points on the bottom plane than between the bottom and top plane. An investigation of the successful MOILPs

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11



(a) Runtime with 2 Objectives. (b) Runtime with 3 Objectives.

(c) Solutions with 2 Objectives. (d) Solutions with 3 Objectives.

Fig. 2. Runtime of MOILP and size of Pareto Front

shows a mean runtime of 19,55 s (compared to 20,49 s in the bi-objective scenario) and a third quantile of just 8.3 s. Thus, if the MOILPs do not fail, they perform comparably good. Unfortunately, it is impossible to predict whether a MOILP will fail without solving it. Thus, the high probability of failure revealed in this analysis renders the approach infeasible for more than 2 objective functions.

The reason for MOILPs running out of memory is the combinatorial explosion of ILPs to be solved. Notably, the size of the Pareto fronts in the 3-objective case is very small as Figure 2(d) depicts. It comprises only up to 30 solutions. The number of ILPs to be solved is much higher, but most ILPs are infeasible. Small systems can have comparably large Pareto fronts (e.g., a generated 3x4 system had a Pareto front with 16 solutions) and large systems can have small Pareto fronts (e.g., a generated 29x24 system had a Pareto front with 4 solutions only). An interesting contrast is the larger size of Pareto fronts in the 2-objective case ($\approx$200) compared to the 3-objective case ($\approx$30). A possible explanation is that the more objective functions exist, the constraint qualifying a solution as being non-dominated (i.e., being part of the Pareto front) is stressed, as such a solution has to be best in all but one objective. The maximum number of solutions in a Pareto front corresponds to the largest objective function (i.e., the objective function comprising the most different values).

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

Investigating 4-objective MOILPs reveals the expected further decline in performance. Here 399 MOILPs (47,44%) run out of memory and 168 MOILPs (19,98%) timed out, where both sets do not intersect. Thus, in total 567 MOILPs (67,42%) failed. The mean runtime is at 85,174 s, which is only slightly worse than for the 3-objective case (79,391 s). Investigating only successfully solved 4-objective MOILPs shows a mean runtime of only 13,11 s and the third quantile of the runtime is also just at 25,75 s. Nevertheless, as shown, 4-objective MOILPs are more probable to fail than to succeed.

5.3 Discussion

In conclusion, bi-objective MOILP are applicable and feasible to compute an optimal system configuration for systems up to 30x30. Unfortunately, using MOILPs with more than two objective functions is more likely to fail than to succeed (the probability of failing is $\approx 60\%$ for 3 objectives and $\approx 70\%$ for 4 objectives, but only $\approx 2\%$ for 2 objectives). Thus, MOILPs with more than two objective functions are applicable, but not feasible to compute an optimal system configuration. In consequence, as each objective function represents an NFP of interest to the user, the presented approach allows to feasibly optimize two NFPs concurrently. More NFPs are theoretically possible, but impractical.

6 Related Work

In [10], de Roo et al. introduce an architectural style (MO2) for software systems as an extension to the component and connector style [11]. According to MO2, the basic elements constituting software are adaptable components (AC), multi-objective optimization components (MOO-C) and transformation components (TC). These three types of components are connected by relations with each other. In comparison to the expressiveness of CCM and QCL, the MO2 style has three major drawbacks: (1) global optimization is hard to express, (2) no means to express contextual dependencies exist and (3) no distinction between software and hardware exists.

In [12], Calinescu et al. present a generic architecture for adaptive service-based systems (SBS). The central constituents of the approach are formal specifications of QoS requirements including the specification of dependencies between QoS requirements, and reasoning techniques, based on high-level, user-specified goals and multi-objective utility functions. In contrast to the approach presented in this paper, the approach by Calinescu et al. applies a-priori multi-objective optimization by combining the different objectives as a weighted sum.

The aggregation of individual objective functions to apply single objective optimization methods has been used in many other approaches, too (e.g., in [13, 14]). Nevertheless, as pointed out by Poladian et al. [3], NFPs with different characteristics cannot be unified. In consequence, a-posteriori approaches to multi-objective optimization as presented in this paper are required.

Final edited form was published in "ICAIS: International Conference on Adaptive and Intelligent Systems. Bournemouth 2014", S. 100 - 109. ISBN: 978-3-319-11298-5
https://doi.org/10.1007/978-3-319-11298-5_11

7 Conclusion

In this paper, an a posteriori approach to multi-objective optimization to determine an optimal system configuration has been presented. We showed how to apply and evaluated Klein and Hannan's approach [7] for this purpose. The evaluation showed the general applicability, but revealed the feasibility for the bi-objective case only. Notably, this assessment is based on the used measurement environment. More powerful resources could render 3- and 4-objective MOILPs feasible in the future. In future work, an investigation of when a subset of all objective functions can be unified should be conducted.

References

[1] Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM TAAS 4, 14:1–14:42 (2009)
[2] Marler, R., Arora, J.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26(6), 369–395 (2004)
[3] Poladian, V., Butler, S., Shaw, M., Garlan, D.: Time is not money: The case for multi-dimensional accounting in value-based software engineering. In: Proceedings of EDSER-5 (2003)
[4] Götz, S., Wilke, C., Richly, S., Püschel, G., Assmann, U.: Model-driven self-optimization using integer linear programming and pseudo-boolean optimization. In: Proceedings of ADAPTIVE, pp. 55–64. XPS Press (2013)
[5] Eastlake, D., Hansen, T.: RFC 6234: US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), IETF Std. (2011)
[6] Beugnard, A., Jézéquel, J.-M., Plouzeau, N.: Contract aware components, 10 years after. Proceedings of Theoretical Computer Science 37 (2010)
[7] Klein, D., Hannan, E.: An algorithm for the multiple objective integer linear programming problem. European Journal of Operational Research 9(4), 378–385 (1982)
[8] Rasmussen, L.M.: Zero-one programming with multiple criteria. European Journal of Operational Research 26, 83–95 (1986)
[9] Eikland, K., Notebaert, P.: LP Solve 5.5 reference guide, `http://lpsolve.sourceforge.net/5.5/` (access on November 26, 2012)
[10] de Roo, A., Sözer, H., Aksit, M.: An architectural style for optimizing system qualities in adaptive embedded systems using multi-objective optimization. In: Proceedings of WICSA/ECSA, pp. 349–352. IEEE (2009)
[11] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., Little, R.: Documenting Software Architectures: Views and Beyond. Pearson (2002)
[12] Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS Management and Optimization in Service-Based Systems. IEEE Transactions on Software Engineering 37(3), 387–409 (2011)
[13] Bratskas, P., Paspallis, N., Kakousis, K., Papadopoulos, G.A.: Applying utility functions to adaptation planning for home automation applications. In: Information Systems Development, pp. 529–537. Springer (2010)
[14] Zeller, M., Prehofer, C., Weiss, G., Eilers, D., Knorr, R.: Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems. In: Proceedings of SASO, pp. 79–88. IEEE (2011)